Partition Detection in Mobile Ad-Hoc Networks Using Multiple Disjoint Paths Set

Michaël Hauspie Jean Carle David Simplot

LIFL/IRCICA - Univ. Lille 1

Abstract

In an ad-hoc network, client-server based application can encounter some problems due to the dynamic topology. One of these problems is when the network splits, leading the server and the client to be physically unreachable from each other. Predicting those partitions could be a very useful feature that can be provided to applications. Indeed, being aware of a future disconnection in the network can help to ensure a better quality of service by adapting the client and/or the server behavior. Algorithms already exists to do this, but they are based on trajectory interpolation which need position information to be provided by a positioning system. This paper propose an original link robustness evaluation method based on the notion of disjoint paths set that allow efficient partition detection without using any kind of positioning system. After showing that the use of disjoint path is relevant for detecting network partitions, we propose a distributed algorithm that collects disjoint paths available between the server and the client and thus show that our partition detection metric can be used in a real network. Moreover, those set of disjoint paths could be used in multipath routing protocols.

1 Introduction

Wireless networks such as Bluetooth [3] or WiFi (Wireless Ethernet IEEE 802.11b) [2] can grant users data access regardless of their location without wired connection.

Nowadays, those networks are mostly used by directly communicating with a base station linked to a wired network and internet. Another application of such technologies are networks based neither on a base station nor any kind of fixed infrastructure. Those networks are useful when no wired link is available such as in disaster recovery or more generally when a fast deployment is necessary. In those applications, mobile computers, or nodes, will communicate by routing messages through the network by multi-hopping protocols [19, 21, 23]. These networks are called MANET for Mobile Ad-hoc NETworks [1].

In that context, client-server based application can encounter some problems due to the dynamic topology. One of these problems is when the network splits, leading the server and the client to be physically unreachable from each other. If connection breaks are real failure in a wired environment, in ad-hoc networks, we must consider it as a normal network behavior because it can occur after a node has moved or a user has turned his device off.

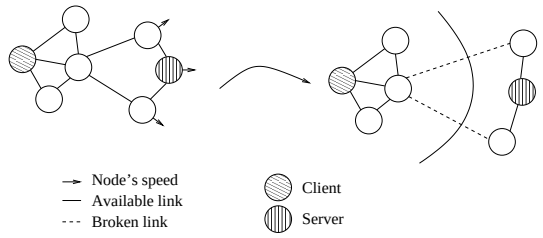


Figure 1: Network partition caused by topology changes

In the case of node movement (Figure 1), it can be useful to predict partitioning and notify applications. Indeed, being aware of a future disconnection in the network can help to ensure a better quality of service

by adapting the client and/or the server behavior. Algorithms already exists to do this but they are based on trajectory interpolation which need position information to be provided by a positioning system. As those systems are often expensive and bulky [4], it may be relevant to provide algorithms that can be used in a "standard" ad-hoc network (*i.e.* with objects that only have a wireless communication device but no positioning system and without centralized infrastructure). In this paper, we propose a metric that can be used to detect partitions based on a multiple disjoint paths set and a distributed algorithm that collects this set using an original directed flooding protocol. The first part of this work has already been presented in [9].

The paper is organized as follows. First, we describe the existing work on partition detection in section 2. Then, section 3 propose and evaluate two metrics based on disjoint paths. Section 4 describe and evaluate our distributed algorithm. Finally we conclude our work in Section 5.

2 Related works

2.1 Partition detection

In [16], Park *et al.* proposed TORA, an adaptive routing protocol for ad-hoc networks. In this protocol, the authors use a method to detect network partitions after they occur. The aim of this detection for their routing algorithm is to find the nodes that are no more reachable and thus, erase the deprecated routes that lead to them. In this paper, we do not focus on detecting partition *after* its occurrence but *before*, so that applications can react by modifying their behavior while the nodes are still connected. Detecting network partition before they happen give some time to seamlessly react by finding a new route and/or adapt the application behavior.

In [20], Shah *et al.* aimed at enhancing data access in an ad-hoc network by detecting partitions before it happens. In their scenario, a node n_1, needs data that are stored in another node (say node n_2). For allowing n_1 to access the data even if its connection with n_2 physically breaks, they propose a data replication mechanism based on partition detection. Every node embeds a positioning system (such as GPS) and by successive measures computes its velocity. Regularly, it spreads those information to the other nodes. Thus, each node of a connected group knows the behavior of the other members of this group. So, they can *predict* when a node storing a particular data will leave the group before it effectively does it. At this point, the owner of the data elects a node of the group to be another host of the data and replicates it on this node. The main advantage of this method is that each node knows exactly *when* the partition occurs if node movements are almost regular (no brutal direction changes). On the other hand, this method has two main disadvantages. First, they need a positioning system, often expensive and bulky. Second, the network has a continuous and relatively high load of the network due to the exchange of position and speed information. Moreover, as the trajectory of nodes are interpolated, it involves some computation that would not be insignificant in the case of small devices such as mobile phones.

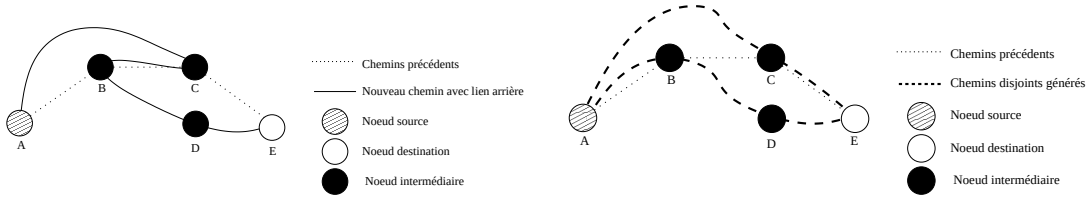
Wang *et al.* proposed a roughly similar partition prediction in [25], although their solution is more centralized. It is based on an extension of the *Reference Point Group Mobility* (RPGM) by Hong *et al.* [10]. They extend this model to handle the velocity of the nodes and thus regroup the node according to their position and speed. For achieving this, each node sends its position and velocity to a server. Then, this server runs a sequential clustering algorithm from the field of pattern recognition [8] to regroup the nodes. Then, as the server knows the groups position and velocity, it can inform nodes of a future partition. Their method has the same problems than those in [20] with one more disadvantage for the need of a centralized server.

Although those algorithms are giving good results, their inputs are depending on expensive and bulky hardware (GPS) or/and on a node that can be reached by the others. It would be preferable to use a method working on every kind of mobile node equipped with wireless device. So, we want to focus our work on a totally distributed network that does not use any positioning system. Indeed, it then can be applied on every ad-hoc network regardless of the hardware or software provided by the nodes. We think that disjoint paths set can also lead to efficient network partition detection as they have good robustness properties. A set of disjoint paths is a set of paths that have no common nodes except the source and the destination. So if a direct link (also called a one hop link) between two nodes breaks, it can discard one and only one path.

Thus, all the other paths are not affected by the failure and the nodes can go on communicating. Moreover, finding disjoint paths set can help developing QoS multipath routing protocols [7, 12, 13, 14].

2.2 Disjoint paths computation

In [15], Haas *et al.* propose a distributed algorithm to compute a disjoint paths set. They computes them iteratively by merging previous paths and an incoming paths that contains *backward* links (*i.e.* links that goes from destination to source, see Figure 2). When a node receives a new path, it check it against the paths it already knows. If a link belongs to both paths, one *forward* (*i.e.* from source to destination) and the other *backward* (*i.e.* from destination to source), two new disjoint paths can be generated by removing the common link and replace it as shown in Figure 2.b where the BC link is the forward one and CB the backward one.



(a) The new path (b) Generated paths

Figure 2: Iterative computing of disjoint paths using backward links

But in the context of our experiments, those kind of path are not used to be generated by standard route request methods and so only a few paths are generated. Actually, route request protocol used by most routing protocols are using *blind* flooding protocols [17, 24]. A sequence number is used to prevent packets to be broadcast twice or more by each node. The problem of this protocol is that backward links can't be generated in a single flood. Indeed, if we look at Figure 2.a., when the first path (ABCE) is generated, node B and C will store the sequence number associated with the route request. If another instance of the route request reaches the node C, it will discard the request because it has already forwarded one with the same sequence number. So, the path ACBDE can't be generated by using a single flood. Thus, to use this method, we need to send multiple route requests. But sending several route requests has two main problems, the time needed to compute the disjoint paths set and the high number of generated packets. A good disjoint paths computation protocol will then be a protocol that slightly increase the number of packets forward in order to generate the highest disjoint paths number but keep an acceptable network congestion.

In [7], Das *et al.* used a flooding protocol no more based on sequence numbers but on the path the packet is currently following. Here is the protocol's algorithm. Let s and d be the source and the destination node respectively. The packet contains a hop count which is decremented at each hop and a route record that store the current path followed by the packet. When a node receives a route request, it performs the following steps :

1. If the node is the destination, a route reply packet is returned to the source along the selected route, as given in the route record that now contains the complete path information between s and d.
2. If maximum hop = 0, the route request packet is discarded.
3. If this node's id[1] is already listed in the route record in the request, the route request packet is discarded to avoid looping.
4. The node's id is appended to the route record of the request packet and this request is rebroadcast.

As the node discards the packet only if there were too many hops or if the path is looping, the protocol is more likely to generate a lot of paths. Indeed, a node can rebroadcast a request more than once. Thus, all loop free paths whose length are less than the hop number used for sending the request are found by this

[1] the node's identity which is a unique number associated with that node

method as only path length and test of previous node in it can avoid rebroadcast. The main problem is that the number of generated packet is very high. Actually, it is clear to see that the complexity of the number of forward for each node is $\mathcal{O}(d^n)$ where n is the number of hops from the source node and d is the density of the network. Thus, this protocol can be used only in small density and short distance (in hop count).

Lee *et al.* extend the blind flooding protocol in [12]. They still using sequence numbers but the decision of a node to forward a packet or not is no more based only on this sequence number. When receiving a packet, a node runs the following algorithm :

1. If the node is the destination, a route reply packet is returned to the source.
2. If a packet with the same sequence number has already been received from the same neighbor, the packet is discarded.
3. The node's id is appended to the route record and the request packet is forwarded.

Here, the complexity of the number of forward for each node will be $\mathcal{O}(d)$. We think that we can lower this while generating disjoint paths using an original flooding method.

In the next section, we present how disjoint paths can be used to detect network partition.

3 Link robustness metrics

We consider the following scenario as the application field of our work. One node of the network is consider as a provider of a given service. Thus, it is acting as server with potentially several clients. Another node is using the service provided by this server, and is the client. We want to give to the client node a method that is able to detect when the server node will be unreachable even by multi-hops routing. To detect this partition, we introduce the notion of link robustness and we propose two metrics based on multiple disjoint paths set to evaluate it. We call link between two nodes the set of paths allowing them to communicate. This link robustness can be seen as its capacity to maintain communication between the two nodes. We will say that a link is strong if the physical disconnection risk is weak.

3.1 Preliminaries

Our evaluation methods are based on neighborhood and paths notions that are to be defined. We use $G(V, E)$ as a graph representing the ad hoc network where V represents the set of wireless mobile hosts and E represents the set of edges. Let v and w be two nodes. A path p between v and w is a series of nodes $o_1, o_2, \ldots, o_n$ such as $o_1 = v, o_n = w$ and for all $i \in [\![1, n[\![,\ o_{i+1} \in N(o_i)$. Let $\tilde{p} = \bigcup_{i=1}^{n} \{o_i\}$, we denote by $|p|$ the number of hops in the path ($|p| = n - 1$). The set of all paths between v and w is denoted by $P(v, w)$.

It is straightforward that each path is not interesting for communications. For instance, extra-long paths or paths with loops are not interesting. Moreover, it is well known that optimal paths in terms of numbers of hops are few, weak and sensible to topological modifications [22]. That is why in this paper, for two given nodes v and w, we will consider a subset of $P(v, w)$ which will be called *sub-optimal loop-free paths*.

Let v and w be two nodes, and $p \in P(v, w)$. p will be called an **optimal** path if and only if $\forall p' \in P(v, w) \quad |p'| \geq |p|$. If p is optimal, $|p|$ is called **distance** between v and w and is denoted by $d(v, w)$. Notice that an optimal path is also a loop-free path[2]. At last, p is called **k-sub-optimal** ($k \geq 0$) if and only if p is loop-free and $|p| < d(v, w) + k$. k represents the number of hops added to an optimal path. We denote by $SOP_k(v, w)$ the set of k-sub-optimal paths between v and w.

Defining SOP_k is motivated by the need of loop-free paths between v and w. Indeed, since optimal paths are few and weak, we need to take account of some other paths near the optimality in number of hops. The set of *Loop-Free-Paths* (denoted by $LFP(v, w)$) can then be determined by

$$LFP(v, w) = \lim_{k \to \infty} SOP_k(v, w) \tag{1}$$

[2] A path $p \in P(v, w)$ is called *loop-free* if and only if $\forall i, j \in [\![1, n]\!]$, $o_i = o_j \Rightarrow i = j$.

For a network of size $|V|$, the longest loop-free path is at most of size $|V|-1$. So, we have $LFP(v,w) = SOP_{|V|-1}(v,w)$. We think that there exists a maximal value of k above which SOP_k does not change significantly in our evaluation. Decreasing k will help us to efficiently compute the metric. Indeed, a fewer k would reduce the network congestion produced by a distributed algorithm computing a set of disjoint paths. Moreover, it is not desirable to store a lot of paths on each node of an ad hoc network.

3.2 Link robustness

Our evaluation method is based on the following idea. The more paths there is between two nodes, the stronger is the path allowing them to communicate. Indeed, if a path breaks, the nodes can go on communicating if and only if it stills being a valid path between them. Figure 3 shows two possible cases.

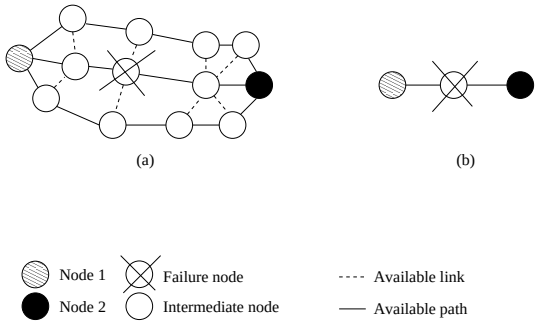


Figure 3: Topology examples

In, Figure 3.a, the nodes have got three paths. If one of them breaks, the nodes can go on communicating. On the other hand, Figure 3.b, it exists only one path connecting node 1 and node 2. If it breaks, the communication will be physically interrupted. In Figure 3 we only consider disjoint paths that seems to be a good robustness criteria as evoked by Haas [15]. Actually, even if it exists hundreds of paths between two nodes, if all of them use the same intermediate node, a simple failure of this node would invalidate all those paths.

For two nodes v and w and a given constant k, the set of parts of $SOP_k(u,v)$ ($2^{SOP_k(u,v)}$) containing only disjoint path containing only disjointed paths is denoted by $DSP_k(v,w)$ (*Disjoint Sub-optimal Paths*) and is defined by :

$$DSP_k(v,w) = \left\{ S \in 2^{SOP_k(v,w)} \mid \quad \forall p,p' \in S \quad \tilde{p} \cap \tilde{p'} \neq \{v,w\} \Rightarrow p = p' \right\}. \tag{2}$$

Thus, if a node disappear, one and only one paths will break. By this fact, as it stills more than one available disjoint path, one can break without physically breaking the connection. Using this notion of disjoint path, we will propose two link evaluation methods.

We consider k as a fixed system parameter for the definition of our metrics so that the notations remain light.

The first one is only based on the number of disjoint paths available. Indeed, if we have only one paths whereas there were more before, it is strongly possible that this one disappears too, resulting to a physical disconnection. This first metric is denoted LR_1 (for *Link Robustness*) and is defined by :

$$LR_1(u,v) = \max_{A \in DSP_k(u,v)} \{|A|\}. \tag{3}$$

For the second one, we aimed at refining the measure given by LR_1. We can actually notice that the longer is a path, the weaker it is. Thus, all paths do not evenly contribute to the link robustness. Then, We evaluate the probability that at least one path composing the link remains available. The set of disjoint paths that gives the best probability that a path survives gives us our second metric denoted LR_2 :

$$LR_2(v,w) = \max_{A \in DSP_k(u,v)} \left\{ 1 - \prod_{p \in DSP_k(v,w)} P_b(p) \right\}, \quad (4)$$

where $P_b(p)$ is the probability that p breaks. The rough calculation of $P_b(p)$ is given by the probability that each direct connection composing the path breaks. If $p = (o_1, o_2, \ldots, o_n)$, we define :

$$P_b(p) = 1 - \prod_{1 \leq i < n} \mu(o_i, o_{i+1}), \quad (5)$$

where $\mu(x, y)$ represents the probability that the direct connection between the nodes x and y remains available. This probability has to be evaluated too. For now, we will consider that μ is a constant.

3.3 Experiments

The aim of this section is to propose and evaluate the metrics. So, we compute our robustness using the global knowledge of the network and generating disjoint paths using a depth first search algorithm. The distributed algorithm used to compute the disjoint paths set will be discussed in Section 4.

To evaluate the metrics, we did the following experiments :

1. a random graph is generated in a rectangular area,
2. we randomly choose a server node and one of its neighbors as the client node[3],
3. nodes move according to the random way-point model [11]. Each node choose a random destination and goes for it. When he reaches it, he waits for a random time, choose another destination and so on,
4. periodically, we compute the link robustness given by the two methods,
5. when a partition occurs, we go back to step 2 until we reached 300 partitions.

We did our experiments at densities of 4, 6 and 8 nodes by communication area. With higher densities, the network is almost always connected so the physical disconnection is not an important problem. With smaller density, the opposite problem occurs, the network is too unstable to predict anything. We used speeds of 1, 1.5 and 2 meters per second which corresponds to different walking speeds. Nodes communication range are set to 10 meters.

Figure 4 and 5 show the evolution of the value computed by both methods. The density used was 8 nodes by communication area. Nodes' speed was 2 meters per second which is a fast walk speed leading to sudden disconnections.

We can observe that robustness values computed by both methods falls just before the physical disconnection. What is observed at this density and this speed is almost the same with other speeds and densities.

To detect if a partition occur, we introduce the notion of threshold. If the metric falls under a given value (called threshold) during a given amount of time (about 1 second in our experiments), we raise a "warning" flag. The relevance of this warning flag will depend on the efficiency of the metric (*i.e.* the number of disconnections it predicted) and on the average time between the prediction and the effective disconnection. If this time is too important, the application would always need to change its behavior according to the warning flag. If this time is too short, the application would not have enough time to changes its behavior and some QoS criterion potentially used by application would not be respected.

So, to achieve evaluating the metrics, we have computed their efficiency and the time spent in alert for a range of thresholds. Figures 6 to 11 show the results for the two metrics. Figures 6 to 9 show how the speed influences on those metrics whereas Figures 10 and 11 show the density's one.

[3]We this, we are sure that the nodes are connected at the start of the simulation

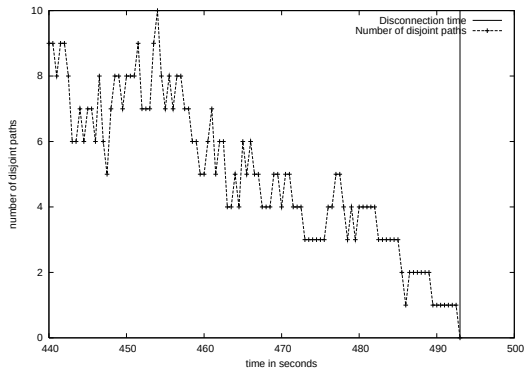


Figure 4: Evolution of the link robustness value for the first metric

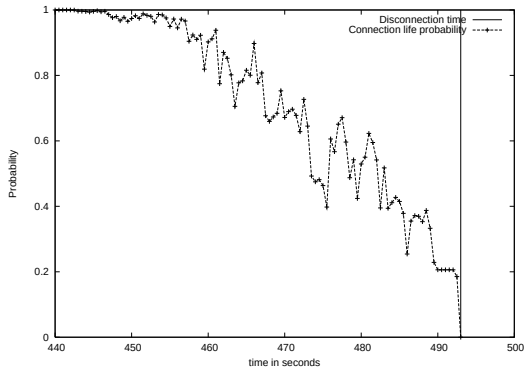


Figure 5: Evolution of the link robustness value for the second metric

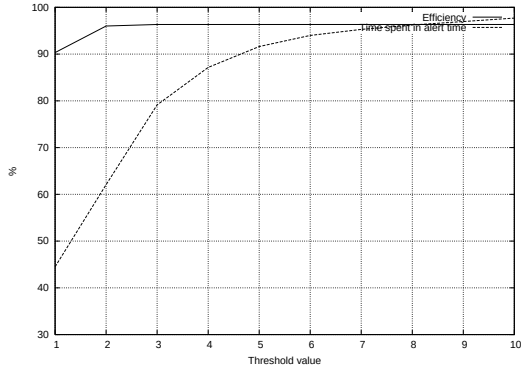


Figure 6: Evaluation of the first metric at 0.5 m/s with density 6

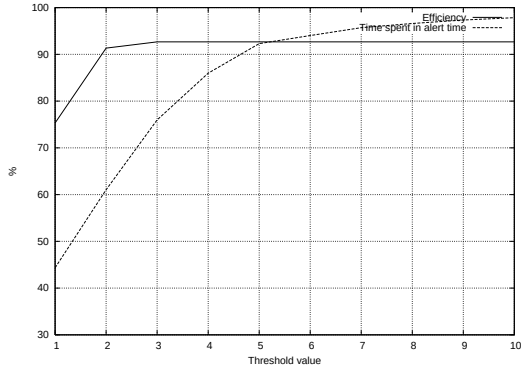


Figure 7: Evaluation of the first metric at 2.0 m/s with density 6

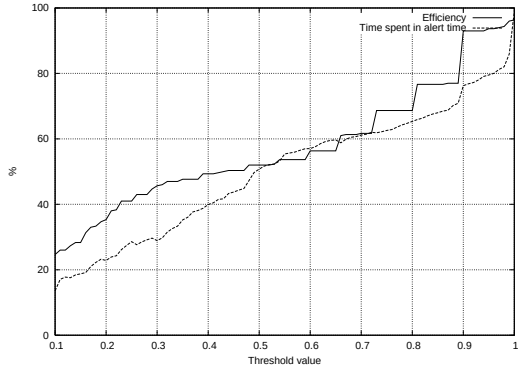


Figure 8: Evaluation of the second metric at 0.5 m/s with density 6

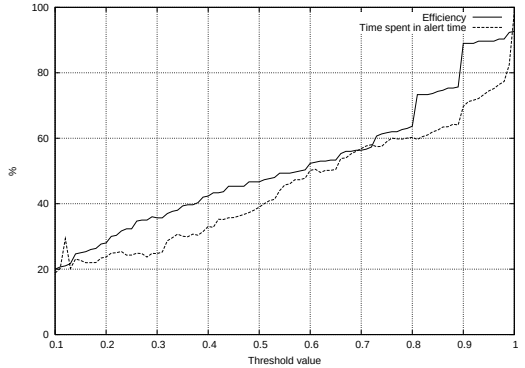


Figure 9: Evaluation of the second metric at 2.0 m/s with density 6

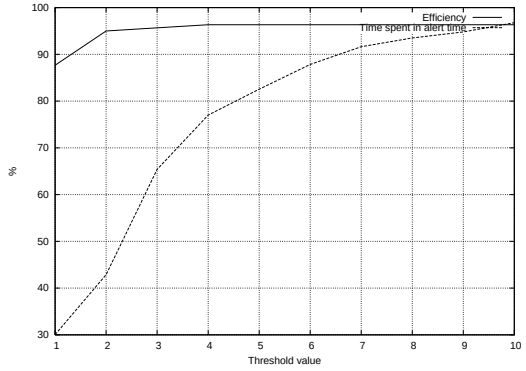


Figure 10: Evaluation of the first metric at 2.0 m/s with density 8

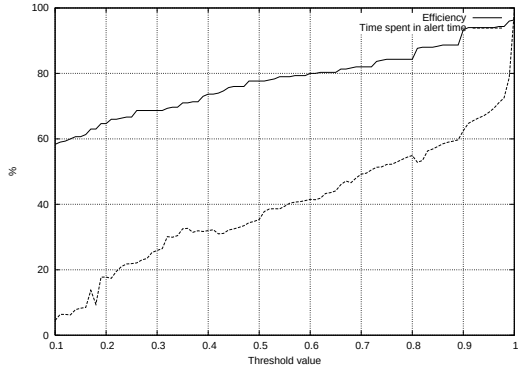


Figure 11: Evaluation of the second metric at 2.0 m/s with density 8

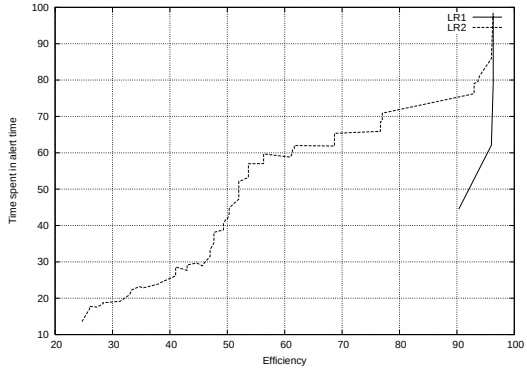


Figure 12: Time spent in alert according to the efficiency (varying threshold)

General observations For both metrics, we can say about speed influence that the faster the nodes move, the worse is the prediction. Indeed, if nodes move faster, the topology is less stable so it is harder to have a good prediction. Concerning the influence of the density, as it grows, the stability of the link grows because the number of disjoint paths is potentially higher. This can be observed in Figures 10 and 11.

Metrics comparisons Bypassing the general observations, we can say that the first metric, that only takes care about disjoint paths, produces far better results. Actually, the gap between the efficiency and the alert time is far more important in the first metric. If we want 90 % of efficiency, in the first metric we can take a threshold of 1 and we will only spent 30 to 50 % of connection time in alert depending on the density and on the speed of the nodes. To reach the same efficiency with the second metric, we would need to spend more than 80 % of the total connection time in alert which is not acceptable and tends to always raise an alert flag. Figure 12 shows in an even better way that LR_1 has a better efficiency/alert time ratio than LR_2 using a density of 6 nodes by communication area and 0.5 m/s speed. This can be explained by the fact that this metric is not as stable as the first one and so, topology modification could cause great fluctuation of its value.

4 Disjoint paths computation

After having shown by our experiments that disjoint paths set provides interesting information which allow efficient partition prediction, we need to find a distributed protocol which computes LR_1. The main problem of the definition of LR_1 is that it is rather impossible to compute its exact value as we need to know all possible disjoint paths leading to the server. For instance, if we consider a client/server link with distance 5, the number of optimal paths (*i.e.* $k = 0$) grows up quickly with the density (see Figure 13).

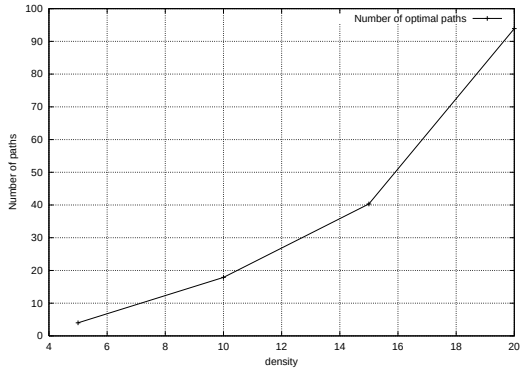


Figure 13: Average number of optimal paths with 5 hops distance

It is clear that finding all paths and compute the subset which maximizes LR_1 requires a lot of network load, memory and CPU time. We must then find an algorithm that gives us an approximative value of LR_1 at low cost in term of network congestion. Moreover, finding disjoint paths set can help developing QoS multipath routing protocols [7, 12, 13, 14].

4.1 Protocol description

Our protocol is based on a blind flooding and improved by two mechanism. The blind flooding has two main problems for our purpose. First, as we said in section 2, it does not generate a lot of disjoint paths because each node forwards the packet once and only once. Second, it floods all the network, even nodes that are not involved. Thus, if we increase the number of forward per node in order to generate more paths, it will jam the whole network.

For the first problem, *Lee et al.* send about d packets per node (where d is the density). The result of this is an increase of the number of paths found but also a high overhead. Our protocol works by *controlling* the maximum of forward per node using multiple sequence numbers.

To solve the second problem, we used hops information to allow only the nodes which are susceptible of being in a path to forward the packet. As a result our request is *directed* towards the destination node and does not overhead the whole network.

Our method works in two steps. First, the client sends a disjoint paths set request to the server. This request can be done using a efficient broadcast [6, 18] or a message sent by a routing protocol [5]. When the server receives the request, he sends a reply back. The reply packet contains the following information :

- the id of the server,
- the id of the client,
- the distance between the server and the client (in hop count). This distance is the one acquired from the request packet,
- the number of hops,
- the current sequence number, used to restrict the rebroadcast of the packet (as in the blind flooding protocol),
- a set of sequence numbers, initially filled by the server,
- the path followed by the packet.

Let $d(c, s)$ be the distance between the client and the server, h the number of hops followed by the packet and $d(c, u)$ the distance between the client and the node u. Every node u receiving this reply runs the following algorithm :

1. if the node has already forward a packet with the same sequence number, the packet is discarded,
2. if $h + d(c, u) > d(c, s) + k$ (where k is a protocol's parameter), the packet is discarded,
3. if the number of hops done by packet is equals to half the distance between the server and client, the node selects a random sequence number in the sequence numbers set included in the packet and rebroadcast the packet using this sequence number else the packet is rebroadcast according to the blind flooding algorithm.

The first rule of the protocol is similar to the blind flooding. The second one is used to *target* the client with the broadcast as shown in figure 14 where thick black lines are the radio link involved by the reply. As we can see, only a few part of the network is disturbed by our protocol.

For this rule, we need to know the distance between every node and the client node. This can be done in several ways. The simplest is to used an optimized broadcast for the request. Each node forwarding the request stores its distance from the client. But it can also be achieved using periodic hello messages (the ones used by a routing protocol for example). The third rule is used to generate a bit more packets and thus, more paths. As the server sets how many sequence numbers to use, it can control the overhead generated by the reply. This overhead can also be controlled by changing the k value. If k is big, there will be more paths and they will be longer. But, on the other hand, there will be more node that will forward the reply and thus, more overhead.

With our sequence number change policy and our limited rebroadcast, we are able to compute an almost large disjoint paths set by generating a small overhead. That is what we show in the next section.

4.2 Experiments

Our experiments were done at several densities using a 200 nodes graph in a 600 meters by 400 meters rectangular area. The nodes' range is adjusted to have the correct density. As in section 3.3, the nodes are following the random way point model. We evaluated one of the two protocols depicted in section 2.2, the one given in [12], and ours. The reason why we do not report the evaluation of the protocol given in [7] is because as we said above, the complexity of this protocol is about $\mathcal{O}(d^n)$ forward per node. Thus, we were unable to reach the end of the simulation for density up to five node per communication area. To have accurate results, we proceed as the following :

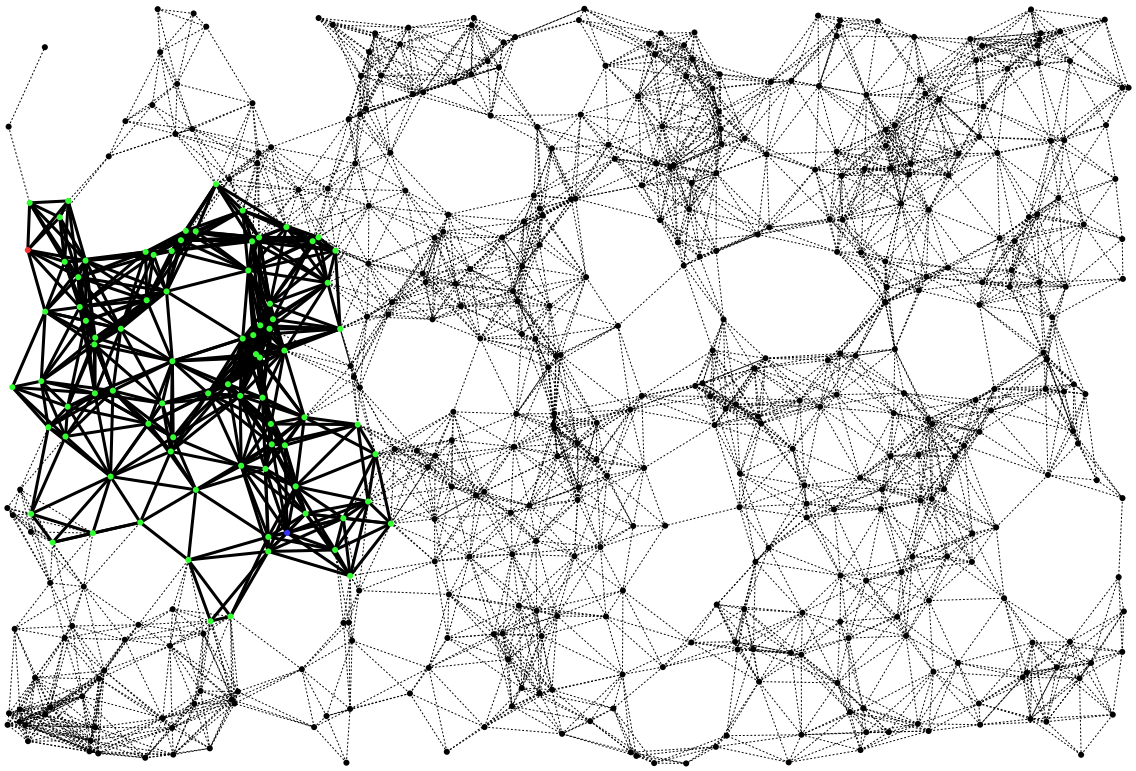


Figure 14: Directed broadcast

1. a random graph is generated,
2. two nodes are selected for being the client and the server. We choose them so that they are at a distance of 5 hops and that there is at least 3 disjoint paths between them (checked by a DFS). If no nodes verifying those conditions are found, a new graph is generated and so on,
3. we simulate the two protocols and store the number of forward of every node and the number of disjoint paths found by each protocols,
4. above steps are repeated for 100 different graphs and the results are averaged.

The k parameter was set to 5 in our protocol and the number of sequence numbers included in the reply packet was equal to the density. Figures 15, 16 and 16 reports the results of our experiments.

Figure 15, reports the number of disjoint paths found by each protocol. As we can see, ours find more than the one depicted in [12].

Figure 16 shows the average number of forward for each node. This represents the amount of overhead in the network. We can notice that what we said in section 2.2 concerning the complexity of *Lee et al.*'s protocol is confirmed by the graph. Indeed, the plot of the number of forward per node is roughly asymptotic to the $f(d) = d$ line. Thanks to our directed broadcast, this number is far beyond for our protocol. Depending on the number of disjoint path we want, we can tune our protocol and thus generate more or less overhead by modifying the k value.

Finally, figure 17 shows the average number of forward per node needed to find disjoint paths. Our experiments shows that our protocol is a real improvement in finding multiple disjoint paths set as the overhead generated is rather small and the number of paths found high.

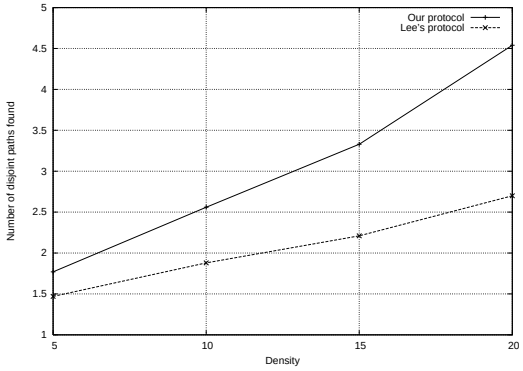


Figure 15: Disjoint paths found

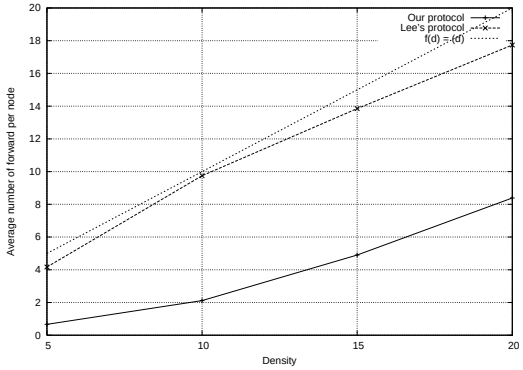


Figure 16: Average number of forward per node

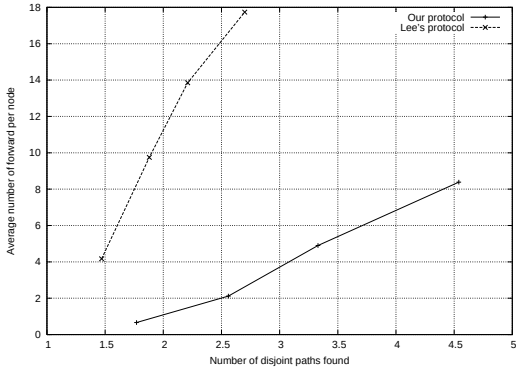


Figure 17: Average number of forward per node over number of disjoint paths found

5 Conclusion

This paper propose two things. At first, we show through our experiments that multiple disjoint paths set can be used to detection network partition between two nodes. Although the prediction could be more accurate, it is a first step to detect the partition without using any kind of infrastructure. This partition detection must rely on an efficient distributed algorithm which can gather as much disjoint paths as possible without jamming the network. We presented such protocol in the second part of the paper and compared it with the ones depicted in the literature. Our experiments shows that our protocol can efficiently find disjoint paths by involving only a small overhead.

References

[1] MANET (Mobile Ad-hoc NETwork) group of IETF (Internet Enginering Task Force). URL: http://tonnant.itd.nrl.navy.mil/manet/.

[2] IEEE standards boards part 11, Wireless LAN medium access control and physical layer specifications, 1999.

[3] Specifications of the bluetooth system v1.0 b, December 1999.

[4] A. Benlarbi-Delaï, D. Simplot, J. Cartigny, and J.-C. Cousin. Using 3D indoor microwave phase sensitive stereoscopic location system to reduce energy consumption in wireless ad-hoc networks. In *Proceedings of the 2nd Smart Objects Conference (sOc'2003)*, Grenoble, France, 2003.

[5] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Mobile Computing and Networking*, pages 85–97, 1998.

[6] J. Cartigny, F. Ingelrest, and D. Simplot. RNG relay subset flooding protocols in mobile ad-hoc networks. *International Journal of Fondations of Computer Science*, to appear.

[7] S.K. Das, A. Mukherjee, S. Bandyopadhyay, D. Saha, and K. Paul. An adaptive framework for QoS routing through multiple paths in ad-hoc wireless networks. *Journal of Parallel and Distributed Computing*, 63:141–153, 2003.

[8] M. Friedman and A. Kandel. *Introduction to pattern recognition: Statistical, Structural, Neural and Fuzzy Logic approaches*, chapter 3, pages 55–98. Imperial College Press, 1999.

[9] M. Hauspie, D. Simplot, and J. Carle. Partition detection in mobile ad-hoc networks. In *Proceedings of the 2nd Mediterranean Workshop on Ad-Hoc Networks*, Mahdia, Tunisia, June 2003.

[10] X. Hong, M. Gerla, G. Pei, and C. Chiang. A group mobility model for ad-hoc wireless networks. In *Proceedings of the 2nd ACM International Workshop on Modeling and Simulation of Wireless and Mobile Systems*, 1999.

[11] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[12] S. Lee and M. Gerla. Split multipath routing with maximally disjoint paths in ad hoc networks. In *Proceedings of the IEEE Internation Conference on Communications*, pages 3201–3205, 2001.

[13] R. Leung, J. Liu, E. Poon, C. Chan, and B. Li. MP-DSR: A QoS-aware multi-path dynamic source routing protocol for wireless ad-hoc networks. In *Proceedings of the 26th IEEE Annual Conference on Local Computer Networks (LCN 2001)*, Tampa, Florida, November 2001.

[14] X. Lin and I. Stojmenović. Location-based localized alternate, disjoint and multi-path routing algorithms for wireless networks. *Journal of Parallel and Distributed Computing*, 2002.

[15] P. Papadimitratos, Z. Haas, and E.G. Sirer. Path set selection in mobile ad hoc networks. In *MobiHoc 2002*, June 2002.

[16] V.D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of IEEE INFOCOM '97*, Kobe, Japan, April 1997.

[17] C.E. Perkins and E.M. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of the Second Annual IEEE Workshop on Mobile Computing Systems and Apllications*, pages 90–100, February 1999.

[18] A. Qayyum, L. Viennot, and A. Laouiti. Multipoint relaying for flooding broadcast messages in mobile wireless networks. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02*, Hawaii, 2002.

[19] E. M. Royer and C-K. Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications Magazine*, pages 46–55, April 1999.

[20] S.H. Shah, K. Chen, and K. Nahrstedt. Cross-layer design for data accessibility in mobile ad hoc networks. In *Proc. of 5th World multiconference on systemics, cybernetics and informatics (SCI 2001), Orlando, Florida*, July 2001.

[21] I. Stojmenović, editor. *Handbook of Wireless Networks and Mobile Computing*. John Wiley & Sons, 2002.

[22] C.-K. Toh. Associativity based routing for ad hoc mobile networks. *Wireless Personal Communications Journal, Special Issue on Mobile Networking and Computing Systems*, 4(2):103–139, March 1997.

[23] C.-K. Toh. *Ad Hoc Mobile Wireless Networks, Protocols and Systems.* Prentice Hall, 2002.

[24] Y.-C. Tseng, S.-Y. Ni, and Y.-S. Chen. The broadcast storm problem in a mobile ad hoc network. *ACM Wireless Networks*, 8(2):152–167, March 2002.

[25] K. H. Wang and B. Li. Group mobility and partition prediction in wireless ad-hoc networks. In *Proceedings of IEEE International Conference on Communications (ICC 2002)*, volume 2, pages 1017–1021, New York City, New York, April 2002.